**http://www.blogjava.net/heavensay/archive/2013/10/20/405440.html**

**一    Prototype、\_\_proto\_\_与Object、Function关系介绍**  
        Function、Object：Js自带的函数对象。

        prototype,每一个函数对象都有一个显示的prototype属性,它代表了对象的原型(Function.prototype函数对象是个例外，没有prototype属性)。

        \_\_proto\_\_:每个对象都有一个名为\_\_proto\_\_的内部隐藏属性，指向于它所对应的原型对象(chrome、firefox中名称为\_\_proto\_\_，并且可以被访问到)。原型链正是基于\_\_proto\_\_才得以形成(note：不是基于函数对象的属性prototype)。

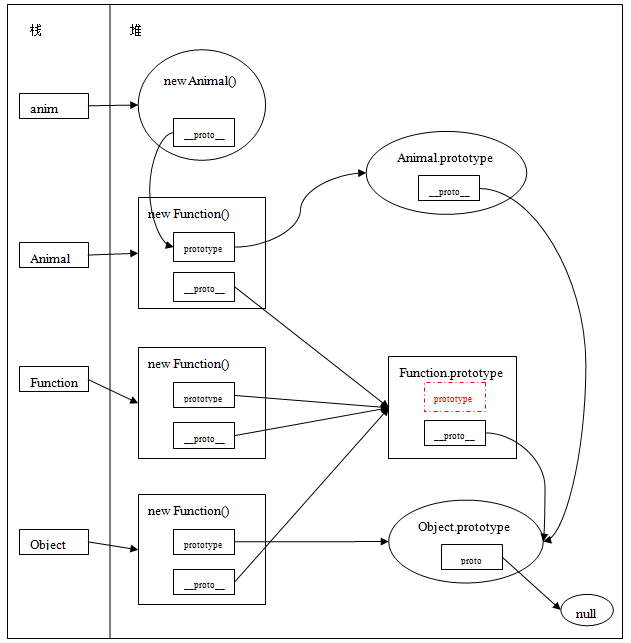
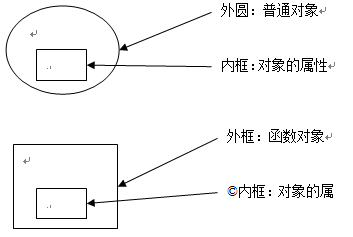
       关于上面提到的函数对象，我们来看以下例子，来说明：

        var o1 = {};  
        var o2 =new Object();  
          
        function f1(){}  
        var f2 = function(){}  
        var f3 = new Function('str','console.log(str)');  
      
        f3('aabb');   // aabb  
        console.log('typeof Object:'+typeof Object);            //function  
        console.log('typeof Function:'+typeof Function);        //function  
        console.log('typeof o1:'+typeof o1);   //object  
        console.log('typeof o2:'+typeof o2);   //object  
        console.log('typeof f1:'+typeof f1);   //function  
        console.log('typeof f2:'+typeof f2);   //function  
        console.log('typeof f3:'+typeof f3);   //function

* 通常我们认为o1、o2是对象，即普通对象；f1、f2、f3为函数。
* 但是其实函数也是对象，是由Function构造的，
* f3这种写法就跟对象的创建的写法一样。f1、f2最终也都像f3一样是有Function这个函数构造出来的
* f1、f2、f3为函数对象，Function跟Object本身也是函数对象。

       Js中每个对象(null除外)都和另一个对象相关联，通过以下例子跟内存效果图来分析Function、Object、Prototype、\_\_proto\_\_对象间的关系。

    function Animal(){  
          
    }  
    var  anim = new Animal();  
      
    console.log('\*\*\*\*\*\*\*\*\*\*\*Animal anim proto\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*');  
    console.log('typeof Animal.prototype:' +typeof Animal.prototype);  //object   
    console.log('anim.\_\_proto\_\_===Animal.prototype:'+(anim.\_\_proto\_\_===Animal.prototype));  //true  
    console.log('Animal.\_\_proto\_\_===Function.prototype:'+(Animal.\_\_proto\_\_===Function.prototype));  //true  
    console.log('Animal.prototype.\_\_proto\_\_===Object.prototype:'+(Animal.prototype.\_\_proto\_\_===Object.prototype));  //true  
      
    console.log('\*\*\*\*\*\*\*\*\*\*\*Function proto\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*');  
    console.log('typeof Function.prototype:'+typeof Function.prototype);  //function  
    console.log('typeof Function.\_\_proto\_\_:'+typeof Function.\_\_proto\_\_);  //function  
    console.log('typeof Function.prototype.prototype:'+typeof Function.prototype.prototype); //undefined  
    console.log('typeof Function.prototype.\_\_proto\_\_:'+typeof Function.prototype.\_\_proto\_\_);   //object  
    console.log('Function.prototype===Function.\_\_proto\_\_:'+(Function.prototype===Function.\_\_proto\_\_)); //true  
  
    console.log('\*\*\*\*\*\*\*\*\*\*\*Object proto\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*');  
    console.log('typeof Object.prototype:'+typeof Object.prototype);  //object  
    console.log('typeof Object.\_\_proto\_\_:'+typeof Object.\_\_proto\_\_);  //function  
    console.log('Object.prototype.prototype:'+Object.prototype.prototype);  //undefied  
    console.log('Object.prototype.\_\_proto\_\_===null:'+(Object.prototype.\_\_proto\_\_===null));  //null  
  
    console.log('\*\*\*\*\*\*\*\*\*\*\*Function Object  proto关系\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*');  
    console.log('Function.prototype===Object.\_\_proto\_\_:'+(Function.prototype===Object.\_\_proto\_\_));   //true  
    console.log('Function.\_\_proto\_\_===Object.\_\_proto\_\_:'+(Function.\_\_proto\_\_===Object.\_\_proto\_\_));   //true  
    console.log('Function.prototype.\_\_proto\_\_===Object.prototype:'+(Function.prototype.\_\_proto\_\_===Object.prototype));   //true  
  
    /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 系统定义的对象Array、Date \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
    console.log('\*\*\*\*\*\*\*\*\*\*\*\*\*\*test Array、Date\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*');        
    var array = new Array();  
    var date = new Date();  
    console.log('array.\_\_proto\_\_===Array.prototype:'+(array.\_\_proto\_\_===Array.prototype));   //true  
    console.log('Array.\_\_proto\_\_===Function.prototype:'+(Array.\_\_proto\_\_===Function.prototype));  //true  
    console.log('date.\_\_proto\_\_===Date.prototype:'+(date.\_\_proto\_\_===Date.prototype));    //true  
    console.log('Date.\_\_proto\_\_===Function.prototype:'+(Date.\_\_proto\_\_===Function.prototype));     //true

**Function、Object、Prototype、\_\_proto\_\_内存关系图**  
  
  
        上面的内存图跟堆栈结构可以参照文章[Javascript\_01\_理解内存分配](http://www.cnblogs.com/fool/archive/2010/10/07/1845226.html)。  
        堆区图说明：  
   
        Function.prototype函数对象图内部表示prototype属性的红色虚框，只是为了说明这个属性不存在。

**通过上图Function、Object、Prototype关系图中，可以得出一下几点：**

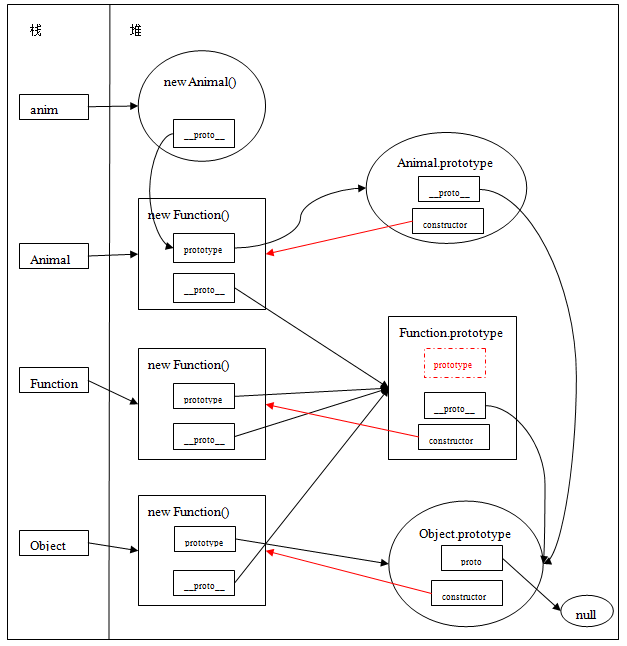
1. 所有对象所有对象，包括函数对象的原型链最终都指向了Object.prototype，而Object.prototype.\_\_proto\_\_===null，原型链至此结束。
2. Animal.prototype是一个普通对象。
3. Object是一个函数对象，也是Function构造的，Object.prototype是一个普通对象。
4. Object.prototype.\_\_type\_\_指向null。
5. Function.prototype是一个函数对象，前面说函数对象都有一个显示的prototype属性，但是Function.prototype却没有prototype属性，即Function.prototype.prototype===undefined，所有Function.prototype函数对象是一个特例，没有prototype属性。
6. Object虽是Function构造的一个函数对象，但是Object.prototype没有指向Function.prototype，即Object.prototype!==Function.prototype。

**二    Prototype跟Constructor关系介绍**

         在 JavaScript 中，每个函数对象都有名为“prototype”的属性(上面提到过Function.prototype函数对象是个例外，没有prototype属性)，用于引用原型对象。此原型对象又有名为“constructor”的属性，它反过来引用函数本身。这是一种循环引用（i.e. Animal.prototype.constructor===Animal）。

        通过以下例子跟内存效果图来分析Prototype、constructor间的关系。

    console.log('\*\*\*\*\*\*\*\*\*\*\*\*\*\*constructor\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*');   
  
    console.log('anim.constructor===Animal:'+(anim.constructor===Animal))    ;    //true  
    console.log('Animal===Animal.prototype.constructor:'+(Animal===Animal.prototype.constructor))    ;    //true  
    console.log('Animal.constructor===Function.prototype.constructor:'+(Animal.constructor===Function.prototype.constructor));   //true  
    console.log('Function.prototype.constructor===Function:'+(Function.prototype.constructor===Function));    //true  
    console.log('Function.constructor===Function.prototype.constructor:'+(Function.constructor===Function.prototype.constructor));    //true  
  
    console.log('Object.prototype.constructor===Object:'+(Object.prototype.constructor===Object));    //true  
    console.log('Object.constructor====Function:'+(Object.constructor===Function));    //true

**prototype、constructor内存关系图(在Function、Object、Prototype关系图上加入constructor元素)：**  


        上图中，红色箭头表示函数对象的原型的constructor所指向的对象。

1. 注意Object.constructor===Function；本身Object就是Function函数构造出来的
2. 如何查找一个对象的constructor，就是在该对象的原型链上寻找碰到的第一个constructor属性所指向的对象。

**参考：**

http://www.cnblogs.com/fool/category/264215.html  (javascript原理介绍)

http://www.libuchao.com/2012/05/14/prototypes-in-javascript/  (JavaScript 的原型对象 Prototype)

http://rockyuse.iteye.com/blog/1426510 (理解js中的原型链，prototype与\_\_proto\_\_的关系)